La librairie graphique p5.js

Introduction

p5.js, c'est quoi ?

- Librairie JavaScript dédiée au « creative coding »
  - Utilisation « simple » grâce à de nombreuses fonctions
  - Graphisme, animations, interactions…
  - Issue de Processing (librairie JAVA)
    - Une page de « computer art » basé sur Processing :
      - http://recodeproject.com/
    - Un tuto pour passer d'un code Processing à du code p5.js
      - https://github.com/processing/p5.js/wiki/Processing-transition

p5.js, comment l'utiliser dans une page web ?

- 2 méthodes pour l'importer dans le fichier .html
  - Télécharge l'archive des fichiers ici et faire un appel local :

    ```
    <script src="js/p5.min.js"></script>
    ```

  - Ou, utiliser un lien en ligne vers la bibliothèque (recommandé pour avoir la dernière version) :
    - Exemple :

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.5.16/p5.min.js">
</script>
```

    - Nécessite une connexion internet
    - Dernière version disponible ici

p5.js : les grands principes (1)

- Le cœur de p5.js est une fonction draw() qui est appelée en boucle, par défaut 60 fois par seconde

```
function draw() {
    // code de la fonction, permettant de générer une image
    ellipse(mouseX, mouseY, 20, 20);
}
```

- C'est le code de cette fonction qui va permettre de **générer une image par programmation**. L'appel en boucle de **draw()** permet de créer une **animation** en faisant évoluer l'image (ex : déplacer des formes). C'est le principe du dessin animé !
  - Remarque : comment changer la fréquence des images ?
    - frameRate(25) : 25 images par seconde

p5.js : les grands principes (2)

- Attention, par défaut les images sont « empilées » les unes sur les autres à chaque appel de **draw() :** les anciens tracés ne s'effacent pas automatiquement
- Pour effacer les tracés précédents, il faut afficher un fond d'image par dessus, au début de **draw()** :
  - Ex : background('white') ; //permet d'afficher un fond blanc
- Remarque : l'affichage d'un fond semi-transparent par dessus les tracés précédents permet de garder une trace éphémère des objets en mouvement
  - Ex : fond noir d'opacité 0.1 en mode couleur RGBA
    - background('rgba(0, 0, 0, 0.1)')

p5.js : les grands principes (3)

- Initialisations
  - Il est souvent nécessaire de faire quelques initialisations avant de lancer en boucle le code d'affichage de la fonction draw().
  - Ces initialisations se font dans une fonction **setup()** appelée au début du programme

```
function setup() {  // code d'initialisation
    frameRate(10) ;
    createCanvas(800,600) ;
}
function draw() {
    // code appelé en boucle
    ellipse(mouseX, mouseY, 20, 20);
}
```

p5.js : les grands principes (4)

- Pré-chargements
  - Lorsqu'on travaille avec des fichiers images (ou autres), il est préférable de les pré-charger avant de lancer le code des fonctions setup() et draw()
  - Ces pré-chargements se font dans une fonction preload()

```
var img ;

function preload(){ // code de pré-chargement

    img = loadImage('https://upload.wikimedia.org/wikipedia/commons/6/6a/JavaScript-logo.png');
}

function setup() {

    frameRate(10) ;

}

function draw() {

    image(img, 0, 0, 100, 100) ;

    ellipse(mouseX, mouseY, 20, 20);

}
```

Dessiner des formes

- Créer une zone de dessin

```
createCanvas(800,600) ;
```

- Rattacher la zone de dessin à un élément du HTML (ex : une div)

```
var myCanvas = createCanvas(800,600) ;

myCanvas.parent('identifiant') ;
```

ou *identifiant* correspond à la valeur de l'attribut *id* de l'élément HTML

Dessiner des formes

- Formes de bases
  - Cercle et ellipse
    - ellipse(x, y, rayonx, rayony)
    - Si *rayonx* est égal à *rayony* on dessine un cercle
  - Carré et rectangle
    - rect(x, y, largeur, hauteur)
    - Si *largeur* est égal à *hauteur* on dessine un carré
  - Triangle :
    - Il faut donner les coordonnées des 3 points du triangle
    - triangle(x1, y1, x2, y2, x3, y3)

Dessiner des formes

- Formes de bases
  - Lignes
    - Il faut donner les coordonnées des 2 points extrèmes de la ligne
    - line(x1, y1, x2, y2)

Dessiner des formes

- Formes personnalisées
  - On déclare le début d'une nouvelle forme puis on décrit les points de passage de la forme (appelés « vertex ») :

```
beginShape() ;
vertex(10, 10) ;
vertex(50, 30) ;
vertex(80, 100)
....
endShape() ;
```

  - Types de formes : paramètre à rajouter dans beginShape (voir la doc en ligne)
  - Fermeture et remplissage des formes : paramètre à rajouter dans endShape (voir la doc en ligne)

Dessiner des formes

- Bordures
  - Par défaut les formes sont dessinées avec une bordure
  - noStroke() : les formes dessinées à la suite n'auront plus de bordure
  - stroke(*couleur*) : rétablit des bordures en définissant leur couleur
    - *couleur* peut être définit de plusieurs façon (RGB, HSC,…) : voir la doc en ligne
    - Le mode de couleur peut être changer avec colorMode()
  - Changer les caractéristiques des bordures :
    - strokeWeight(), strokeCap(), strokeJoin()

Dessiner des formes

- Remplissage des formes
  - Par défaut les formes sont remplies avec un fond blanc
  - noFill() : les formes dessinées à la suite n'auront plus de remplissage (fond transparent)
  - fill(*couleur*) : permet de changer de couleur de remplissage
    - *couleur* peut être définit de plusieurs façon (RGB, HSC,…) : voir la doc en ligne

Dessiner des formes

- Transformations
  - Translation
    - translate(*x0, y0*) : pour toutes les formes dessinées à la suite, le point d'origine est modifié en *x0* et *y0*
  - Rotation
    - rotate(*a*) : toutes les formes dessinées à la suite seront tournées d'un angle *a*
      - l'unité d'angle peut être configurée grâce à angleMode() (en radians par défaut)
  - *Remarque :*
    - *Ces transformations sont remises à zéro à chaque appel de draw()*
    - *Elles se cumulent si on les appelle plusieurs fois dans draw()*

Dessiner des formes

- Mode de tracé des rectangles
  - Permet de définir :
    - Point du rectangle qui sert de référence pour son placement
    - La signification des dimensions (largeur ou demi-largeur, hauteur ou demi-hauteur)
  - Ex : coin haut gauche (mode part défaut : CORNER)

    centre du rectangle : rectangleMode(CENTER)

    …

Dessiner des formes

- Utiliser un contexte temporaire de dessin
  - permet de définir temporairement une translation, rotation, couleurs, etc. pour un ensemble de formes
    - push() ; //début du nouveau contexte de dessin

      fill('black') ;

      rotate(60) ;

      translate(100, 100) ;

      ellipse(0,0, 20, 20) ;

      …

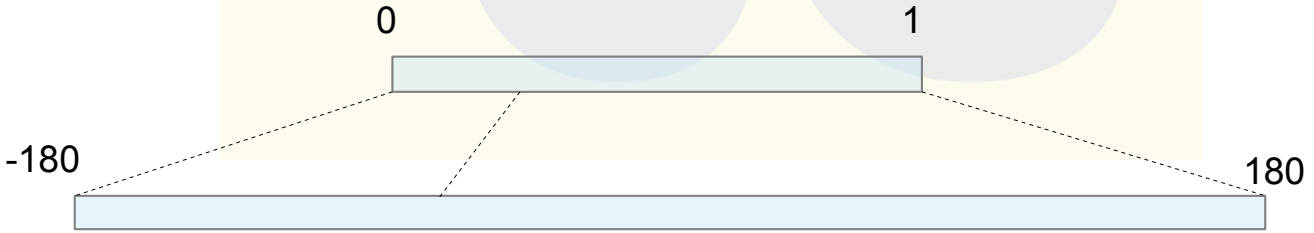
      pop() ; // fin du nouveau contexte de dessin et retour au contexte précédent

Dessiner des formes

- Un peu d'aléatoire
  - random(*[min]*, *[max]*)
    - Renvoie un nombre décimal aléatoire uniformément réparti entre :
      - random() : 0 (compris) et 1.0 (non compris)
      - random(*max*) : 0 (compris) et *max* (non compris)
      - random(*min, max*) : *min* (compris) et *max* (non compris)
  - random(*tableau*)
    - Renvoie un élément aléatoirement tiré dans *tableau*
  - noise(*x*)
    - Renvoie un nombre aléatoire entre 0.0 et 1.0 de type « bruit de Perlin » (bruit naturel, plus « doux » que random())
    - *x* doit évoluer entre chaque appel de noise()
      - Evolution importante (+ 0.1) : bruit « rapide »
      - Evolution faible (+ 0.01) : bruit « lent »

Dessiner des formes

- Changement d'échelle (linéaire)
  - map(*value, fromMin, fromMax, toMin, toMax, [withinBounds]*)
    - Permet de transformer une valeur value depuis une échelle définie par une valeur min *fromMin* et une valeur max *fromMax*, vers une échelle avec une valeur min *toMin,* et une valeur max *toMax*
    - *WithinBounds* est un booléen (valeur True ou False) indiquant si l'on doit contraindre la valeur retournée au bornes min et max de l'échelle d'arrivée
    - Ex : map(noise(x), 0, 1, -180, 180, True)



Quelques constantes utiles de p5.js

- *width* et *height* : largeur/hauteur de la zone de dessin (canvas)
- *windowWidth* et *windowHeight* : largeur/hauteur de la fenêtre de navigateur
  - Peut servir à redimensionner la zone de dessin lorsque la fenêtre est redimensionnée, en utilisant la fonction windowResized()
- *displayWidth* et *displayHeight* : largeur/hauteur de l'écran
  - Peut servir à faire un sketch plein écran
- *mouseX* et *mouseY* : contient en permanence les positions en X et Y de la souris

Animer des formes

- Notions de mouvement
  - Mouvement rectiligne uniforme
    - La trajectoire est une **droite** et la **vitesse est constante**
    - Le déplacement de la forme (en *x* et en *y*) entre chaque tracé de draw() est toujours le même car le temps entre chaque appel est constant (fixé par *frameRate()*)

Animer des formes

- Notions de mouvement
  - Mouvement rectiligne uniforme
    - Exemple :
      - *x : variable contenant la position en x d'une « balle »*
      - *y : variable contenant la position en y d'une « balle »*

      ellipse(x, y, 30, 30) ; // tracé de la balle en x et y

      x = x + 10 ; //déplacement du centre de la balle en x pour le prochain tracé

      y = y + 30 ; //déplacement du centre de la balle en y pour le prochain tracé
    - Remarque : ici 10 et 30 forment le vecteur vitesse de la balle

Animer des formes

- Exercice
  - Flocons de neige
    - Animer des flocons de neige (cercles blancs sur fond noir) :
      - Position de départ et taille
        - Y : 0 (haut)
        - X : aléatoire
        - taille (rayon) : aléatoire entre 5 et 10 px
      - Vitesses :
        - VX : aléatoire entre -25 et 25 px
        - VY : aléatoire entre 0 et 1 px
    - On créera un nouveau flocon toutes les 500 millisecondes avec la fonction setInterval() de Javascript (à placer à la fin du programme):
      - setInterval(maFonction, 500) ; // la fonction « maFonction » est appelée toutes les 500 millisecondes

Animer des formes

- Notions de mouvement
  - Mouvement accéléré uniformément
    - L'**accélération est constante** (ex. : la gravité) : la vitesse augmente suivant la direction de l'accélération
    - L'augmentation de la **vitesse** entre chaque tracé de draw() est toujours la même (temps constant)

Animer des formes

- Notions de mouvement
  - Mouvement accéléré uniformément
    - Exemple :
      - *x : variable contenant la position en x d'une « balle »*
      - *y : variable contenant la position en y d'une « balle »*
      - *vx : variable contenant la vitesse en x de la « balle »*
      - *vy : variable contenant la vitesse en y de la « balle »*

      ellipse(x, y, 30, 30) ; // tracé de la balle en x et y

      vx = vx + 3; //augmentation de la vitesse en x pour le prochain tracé

      vy = vy + 5 ; //augmentation de la vitesse en y pour le prochain tracé

      x = x + vx ; //déplacement du centre de la balle en x pour le prochain tracé

      y = y + vy ; //déplacement du centre de la balle en y pour le prochain tracé
    - Remarque : ici 3 et 5 forment le vecteur accélération de la balle

Animer des formes

- Notions de mouvement
  - Mouvement circulaire uniforme
    - La trajectoire est un cercle et la **vitesse angulaire** (vitesse de rotation sur la trajectoire) est constante
    - Il faut raisonner en **coordonnées polaires** : *r* est le rayon du cercle de la trajectoire et *a* l'angle de rotation sur la trajectoire
    - L'angle de rotation *a* change d'une même valeur entre chaque tracé de draw()
    - Pour tracer la forme il faudra traduire les coordonnées polaires (r, a) en coordonnées cartésiennes (x, y)
      - **x = r*cos(a)** si a est en radians, **x = r*cos(a*PI/180)** si a est en degrés
      - **y = r*sin(a)** si a est en radians, **y = r*sin(a*PI/180)** si a est en degrés

Animer des formes

- Notions de mouvement
  - Mouvement circulaire uniforme
    - Ex :
      - r : rayon de la trajectoire en pixels
      - a : angle de rotation sur la trajectoire en degrés
      - x0, y0 : position du centre de la trajectoire

      x = r*cos(a*PI/180) + x0 ;

      y = r*sin(a*PI/180) + y0 ;

      ellipse(x, y, 30, 30) ;

      a = a +10 ; // changement de l'angle de 10 degrés pour le prochain tracé

Animer des formes

- Notions de mouvement
  - Mouvement en spirale
    - Il suffit de reprendre le mouvement circulaire et de faire diminuer le rayon jusqu'à 0
      - Ex :
        - va : vitesse angulaire
        - vr : vitesse de décroissance/croissance du rayon

```
x = r*cos(a*PI/180) + x0 ;
y = r*sin(a*PI/180)  + y0 ;
ellipse(x, y, 30, 30) ;
if(r < vr ||  r>300){
      vr = -vr ;  // on inverse le sens de vr
      va = -va;  // on inverse le sens de va
      }
a = a +va;
r = r+ vr ;
```

Animer des formes

- Exercice : spirale chromatique

1) Dessiner une spirale avec changement de couleur suivant l'angle
2) Faire tourner cette spirale à vitesse constante
3) Ajouter des lignes dont les centres suivent la trajectoire de la spirale :
    - Longueur proportionnelle au rayon de spirale (r/4)
    - Angle d'inclinaison proportionnel à l'angle de spirale (essayer plusieurs valeurs)
    - Couleur suit la couleur de la spirale
4) Rejoindre les extrémités de ces lignes successives
5) Faire varier l'angle des lignes en fonction de la position de la souris

Animer des formes

- Notions de mouvement
  - Mouvements avec perte d'énergie (frottements) :
    - Pour simuler une perte d'énergie due à des frottements, on multipliera les vitesses par un facteur constant légèrement inférieur à 1 (la valeur 1 correspondant au cas idéal sans frottement). On ajoutera un « seuil » pour arrêter complètement l'objet si la vitesse devient trop petite
    - Ex :

```
if ( sqrt(vx*vx+vy*vy) > 2){
    vx = 0.98*vx ;
    vy = 0.98*vy ;
}
else{
    vx = 0 ;
    vy = 0 ;
}
```

Animer des formes

- Rebonds
  - Exemple : rebond contre un « mur » droit vertical (axe y)
    - Rebond parfait :
      - On inverse la vitesse en x :

        vx = -vx
    - Rebond avec perte d'énergie :
      - On inverse la vitesse en x avec un facteur multiplicatif inférieur à 1 :

        vx = -0.98 * vx